Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 5 (2011) 593–600

Procedia Computer Science

The 8th International Conference on Mobile Web Information Systems (MobiWIS)

An Adaptive Context-Aware and Event-Based Framework Design Model

Eduardo S. Barrenechea, Paulo S. C. Alencar

{ebarrene,palencar}@cs.uwaterloo.ca
David R. Scheriton School of Computer Science
University of Waterloo

Abstract

Context-aware and proactive technologies have been continuously used over the past years to improve user interaction in areas such as searching and information retrieval, health care and mobile computing. Although there have been significant advances in context-aware systems, there is still a lack of approaches that model and implement context-aware proactive applications involving the combination of context and distributed events. In this paper we address these issues by defining a context-aware event model, a new context-aware publish subscribe scheme and a distributed event-based framework. Our proposed event model is implemented as a context-aware distributed event-based framework that provides the necessary infrastructure to publish and deliver events based on a component's context. In summary, we are able to leverage context as part of our event model and bring behaviour context-aware adaptation to publication and subscription of events.

Keywords: Context-Aware Frameworks, Distributed Event-Based Systems, Mobile Computing

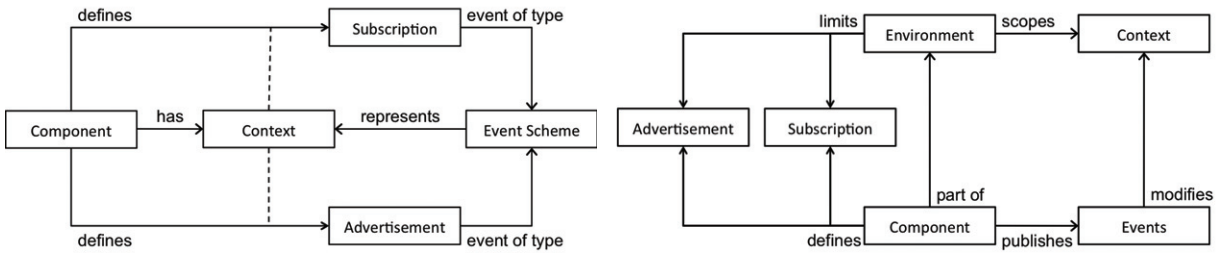
1. Introduction

Context-aware and proactive technologies have been continuously used over the past years to improve user interaction in areas such as searching and information retrieval, health care and mobile computing. With the introduction and increased use of smartphones and other mobile computing devices such as netbooks and tablets, context-aware applications have become potentially able to use the environment's information to provide real-time and autonomous adaptation to the user and his context.

Although there have been significant advances in context-aware systems, there is still a lack of approaches that model and implement context-aware proactive applications involving the combination of context and distributed events. First, existing context-aware systems do not rely on more efficient context information dissemination schemes provided by distributed event-based systems. Second, distributed event-based models and approaches do not use context as a first class element. Third, current publish subscribe approaches that use context only consider the context of external components and not its local context. For example, the notification of an event to a subscriber should not depend on context contraints determined by the publisher and a subscriber should be oblivious to the publisher's context when subscribing to an event. Fourth, there is a need to separate and decouple context information from application logic.

In this paper we address these issues by defining a context-aware event model, a new context-aware publish subscribe scheme and a distributed event-based framework. The context-aware event model introduces context as a first class element. It allows a component to specify the context that relates to its advertisements and subscriptions.

Selection and/or peer-review under responsibility of Prof. Elhadi Shakshuki and Prof. Muhammad Younas.
doi:10.1016/j.procs.2011.07.077



(a) Publish Subscribe View (b) Environment View

Figure 1: Context-aware event model.

Our publish subscribe scheme takes such relationships into consideration through the use of context filters. Context filters are specified by a component to identify under which circumstances it should be notified of or allowed to publish a certain event. By allowing a component to control its different publication and subscription behaviours, we can effectively separate context information from the component logic and place it on the publish subscribe scheme. Our proposed event model is implemented as a context-aware distributed event-based framework that provides the necessary infrastructure to publish and deliver events based on a component's context. In summary, we are able to leverage context as part of our event model and bring behaviour adaptation to publication and subscription of events.

2. Context-Aware Event Model

Our proposed approach consists of introducing context as a first class element into the model. This requires an explicit representation of both syntax and semantics and how context interacts with both events and components. Figure 1(a) shows the publish subscribe view of the event model, while Figure 1(b) shows the environment view of the event model. An *event* is a data representation of a happening in the system. All events have an *event schema*. An event schema specifies the data attributes all events implementing the event schema must have. The name of an event schema is assumed to be unique.

The system supports the following main functions, namelly (un-)subscription, (un-)advertisement, and publication, and is composed of independent, self-contained software entities called *adaptive components* (components for short). Components are uniquely identified and each component executes in its own program space. As shown on Figure 1(a), components have the ability to publish and/or subscribe to events. All context sources in the system are components. A component that is a context source should incapsulate all of the necessary modules to capture and process the context information. All context sinks in the system are components. A component that is a context sink subscribes to the event schema corresponding to the context information it requires.

Not all components are context aware, and not all context-aware components are context sources or sinks. This distinction is very relevant since a component can still be context aware, i.e., adapt its behaviour to the current context state, without the need of keeping track of the context information. Context awareness is accomplished transparently and autonomically through the use of context filters in publications and subscriptions. An event will either be published by the system or delivered to a given component if and only if the context filtering expression matches the context of the environment. A component can achieve behaviour adaptation by controlling under which context it is notified of an event or under which context an event it publishes should be valid.

Such approach removes the burden of determining the validity and applicability of the event notification or publication from the component and places it on the system. A component is no longer in need of providing its own copy of the context model or contacting an external context database. It is also not required to keep track of any context changes in the environment. With context aware adaptation being part of the publish subscribe scheme in the form of context filtering expressions, a component is effectively a collection of behaviours that are combined provide a specific functionality. All of the application logic is contained inside the component and clearly differentiated from context information.

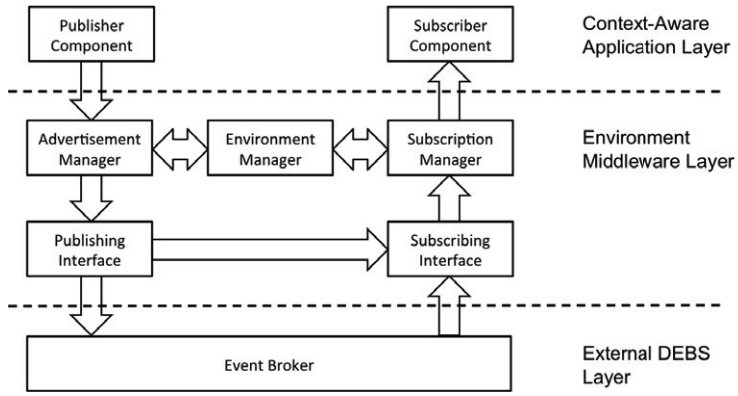


Figure 2: Architecture of the context-aware event-based framework.

As shown in Figure 1(b), components are grouped into *contextual environments* (environments for short). An environment is a logic region used to set bounds to an instance of the context model. All of the components inside the same environment are considered siblings. An environment is able to scope context by allowing sibling components to share the same context parameters and values. A component can be part of more than one environment at a time. Subscription and advertisement of events belong to a single environment. If a component belongs to more than one environment its subscription and advertisements are distinct between environments.

Any event that affects the values of context parameters in an environment will also affect the context of the components contained inside the environment. Context filtering in both event advertisement and subscription is based on the context parameters of the environment the component is part of. An event will only be published to sibling components and/or propagated to the event broker if the context filter expression specified in the advertisement matches the context parameters of the environment. In the case that the event being published carries an update to the context of an environment, the update takes precedence over context filter expression evaluation. Similarly, an event will only be delivered to a component if the context filter expression specified in the subscription matches the context of the environment.

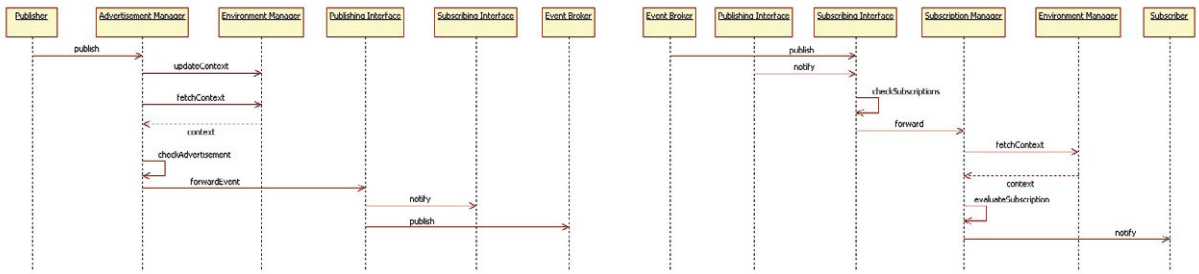
3. Context-Aware Event-Based Framework

The architecture of the framework is shown on Figure 2. The framework is composed of three different layers, the context-aware application layer, where custom context-aware components and application are placed. These components are the publishers and subscribers in an environment. The middleware layer contains the administrative components responsible for managing advertisements, subscriptions along with content and context filtering. The bottom layer refers to the underlying distributed event-based system used to support our system. Figure 3(a) shows the process of publishing an event and Figure 3(b) shows the process of receiving an event.

3.1. Context-Aware Application Layer

All different custom components in the system are placed in this layer. A component may relate to an entire self-contained application or a simple module or service providing some specific functionality. Components are responsible to all generation and consumption of events in the system. Components can be either context providers or context consumers. A context provider is a component that generates context events and propagates them through the system, either inside or outside the environment. Context consumers are components that use context information in order to provide behaviour adaptation. Events without context can also be propagated to environments and components throughout the system. Components can be classified into plain components and context-aware components.

A component is considered plain if it is completely oblivious to context. In other words, plain components do not subscribe to context events or use any form of context filtering in their subscriptions. Plain components can still be context sources and publish context events. Being able to sense some type of context, in other words, having



(a) Publishing an event. (b) Receiving an event.

Figure 3: Publication and subscription sequence diagram.

knowledge of some type of context is not sufficient to characterize context awareness. Both knowledge and adaptation are required for a component to be considered context-aware.

A component is considered context-aware if and only if it has knowledge of some type of context and if it is able to change its behaviour according to changes in said type of context. A context-aware component doesn't necessarily need to receive context information by subscribing to context events. Knowledge of context can be expressed through context filters in its subscription. Behaviour adaptation follows from specifying under which circumstances it is to be notified of a specific event. This approach guarantees that a component is aware of some type of context and that changes to this type of context may affect its behaviour. Any component that uses context filters in theirs subscriptions are considered context-aware.

3.2. Environment Middleware Layer

The environment middleware layer contains many different administrative components that are used to enable advertisement, publication and subscription to take place in the system. It is also responsible for context filtering and communication with other peers or event brokers in a distributed event-based system.

The *environment manager* is the administrative component responsible for keeping track of the state of context in the environment. It has its own instance of the context model which is updated according to event published both from inside and outside the environment. The environment manager is responsible for context disambiguation. It provides the necessary logic to analyze context events, extract the context information and ensure that the context model instance is always in a coherent state. The environment manager is also accessible to components in the context-aware application layer. This access provides a simple query interface service that can be used by components to access the context information in a way that is similar to a context knowledge base. The environment manager is not limited only to the role of a context repository. It can contain other context-related components such as context interpreters and context reasoning capabilities that are used to extract higher-order context information from the context model.

The *advertisement manager* is responsible for handling all context filtering in advertisements from components in the environment. When a component publishes an event, the advertisement manager checks if the event has any context information pertaining to the environment. The environment manager is notified of any context information present in events being published from inside the environment. After the context state is updated, the advertisement manager processes the event publication according to the parameters specified in the components advertisement of the event. The advertisement manager checks if the event matches any event schema in the component's advertisements, and if the current context state matches the specified context filtering expression. The event is forwarded to the publishing interface if those conditions hold.

The *subscription manager* is responsible for handling all context filtering in subscriptions from components in the environment. It also forwards context information from events originating outside the environment to the environment manager. Such events may still affect the context state of the environment and should be considered by the environment manager. The subscription manager is able to match components with their subscriptions, request context information from the environment manager and evaluate context filtering expressions. An event is delivered to a subscriber if

the event schema matches the one specified in the subscription, and if current environment context state matches the context filtering expression.

The *publishing interface* is an event forwarding component. It handles dissemination of both private and public events. The publishing interface contains all of the necessary logic to publish an event both to the environment's subscribing interface and to any underlying distributed event-based system. The *subscribing interface* is an event aggregator component. It aggregates all events directed at the environment, including events originating from inside the environment. The subscribing interface is responsible for issuing and updating subscriptions to the underlying distributed event-based system. Outside events are forwarded to an environment if and only if a subscriber component explicitly states the correct privacy scope in its subscription to an event schema.

3.3. DEBS Layer

The DEBS layer refers to the external distributed event-based system to which this environment is connected to. Such systems can be other distributed event-based middleware systems, which can be represented by an event broker node, or other environments connected in a peer-to-peer fashion. Such approach is very flexible in terms of network layout and control.

4. Scenario Based Evaluation

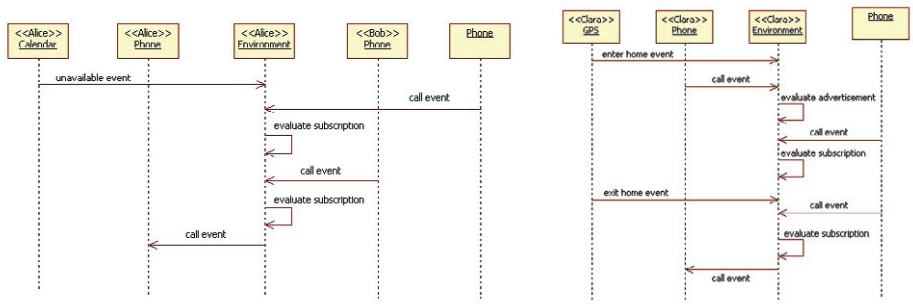
In this section, we provide a scenario-based evaluation [1] of our context-aware distributed event-based framework by walking through different scenarios and demonstrating how different parts of the system work together. For our sample use cases, we consider the realm of mobile devices. The various applications in a mobile device can relate to different components in our distributed event-based system. Each component has its own function, for example, the Phone component is responsible for making and receiving calls, the AddressBook component holds the user's contacts information, the Calendar component holds the user's schedule and the GPS component is responsible for keeping track of the user's location.

Context in mobile devices is very dynamic since these types of devices are constantly with the user, changing environments according to the user's actions. Mobile devices are also very constrained in terms of memory, processing power and sometimes may be very hard to use when a lot of user input is required. Processing power and memory constraints encourage the different components to focus on providing a single functionality instead of being all encompassing applications like some desktop applications. Context awareness can be achieved simply by providing a clean and effective way to share information between these different components. To illustrate the applicability of the context-aware publish subscribe scheme, we have relied upon a simple advertisement and subscription representation based on expressions such as *[subscription|advertisement](EventSchema, Context-Aware Filtering Expression)*. The event schema represents occurrence of events such as incoming or outgoing calls. Filtering expressions are logic expressions involving context variables, e.g., $variable_1 = value_1 \vee variable_2 = value_2$.

4.1. Use Case: Scheduled Meeting

Suppose in our use case that Alice is in a meeting with her co-workers and only wants to receive calls from Bob, her boss. Current approaches in smartphones do not allow for easy automation of such scenario. Alice needs to be aware of her schedule and switching her phone to silent upon entering a meeting. Even if this is the case, Alice still needs to check any calls she might receive during her meeting and manually screen the calls from Bob. Such simple adaptation can be easily achieved through our approach.

The pieces of context that are represented in our scenario are Alice's status, i.e. *unavailable* when in a meeting and *available* otherwise, and her relationship with Bob, i.e. calls from her boss are important to her. Alice's smartphone Calendar component is responsible for publishing events relating to her status whenever there's a change in her schedule. By creating the following subscription, the Phone component can be made aware of Alice's status and adapt its behaviour to ring only if she is available, *subscription(IncomingCall, Status='available')*. If Alice's status is *unavailable* then the subscription to incoming calls is not valid and the *IncomingCall* event will not be propagated to the Phone component. When Alice's status is *available* then the subscription is valid and the *IncomingCall* event gets propagated. In the case of allowing Bob's call to go through, another subscription *subscription(IncomingCall, Caller='Bob')* can be appended. If either of the subscriptions is valid the event will be propagated to the Phone



(a) Scheduled meeting use case sequence diagram.

(b) Context-aware calls use case sequence diagram.

Figure 4: Use case sequence diagrams.

component. The same result can be achieved using a single subscription that combines both filtering expressions *subscription(IncomingCall, Status='available' ∨ Caller='Bob')*. As a result, even if Alice's status is *unavailable* she can still receive Bob's call. Figure 4(a) shows the sequence diagram for this use case.

4.2. Use Case: Context-Aware Calls

Suppose now that Alice doesn't want her teenage daughter Clara to use her cell phone at home. Currently there's no simple way to guarantee such functionality other than by taking the device away from Clara. With our approach it becomes a matter of having the correct subscriptions and publication expressions.

Clara's current location is provided by the GPS component in her cell phone. Whenever Clara's location changes, the GPS component publishes a *LocationChange* event that results in the Context Broker updating the context knowledge base for this environment. By using the following advertisement expression in the Phone component *advertisement(OutgoingCall, Location!='home')* we can prevent Clara from making outgoing calls when at home. If the Phone component publishes an *OutgoingCall* event the Context Broker checks its current location and validates the context state of the mobile device. If the location is different from home then the *OutgoingCall* event gets propagated to the service provider, requesting a connection to the target phone. If the location is home, this event publication is not propagated and the outgoing call connection will not be requested. Similarly, by adding the subscription *subscription(IncomingCall, Location!='home')* we can prevent Clara from receiving incoming calls while at home. Figure 4.2 shows the sequence diagram for this use case.

5. Discussion

In previous sections we have presented the primary building blocks of an adaptive context-aware distributed event-based framework. Our insights with scenario-based evaluation have unveiled three design issues, which are discussed next.

A. Context availability. As discussed in the use case described in Section 4(a), the first design concern is related to context availability. In current context-aware approaches, a component must have access to a context source that provides the information it requires in order to adapt its behaviour to the user's context. In other words, Alice's Phone component is required to have access to context information being provided by Alice's Calendar component. A proposed solution using current context-aware frameworks would require the Phone component to search for a schedule context provider and register with it to receive updates to Alice's schedule. This component would be Alice's Calendar component. The Calendar component would contact the Phone component whenever a change in Alice's schedule takes place. Alice's Phone component would be required to deal with said context information on the source code level, that is a hardcoded solution. This affects the overall design and implementation of the Phone component. It is now required to keep track of schedule changes, which is not it's intended purpose. Whenever the Phone component receives a call notification, it is then required first check if it's stored status allows it to ring.

B. Incremental Context-Aware Publications and Subscriptions. A second design concern relates to extending context-aware publications and subscriptions. As seen in Section 4(a), Alice's Phone component should not be dependent only on Alice's status, but also depend on the callers identity. Alice's Contacts components holds all the information relating to the people close to Alice and their relationship with her. If Alice's status is unavailable the Phone component is required to check if the caller is allowed to go through. In order for Bob's call go through, the Phone component has to access Alice's Contacts component. This once again requires changes to the Phone component and adds extra functionality that does not relate to the Phone component's main objective of placing and receiving calls. Such hardcoded solution is in clear contract with the straightforward approach provided by our context-aware publish subscribe scheme. Notice that the proposed context-aware publish subscribe scheme does not only support receiving events based on context changes, but also supports filtering of events that can be received based on contextual subscriptions.

C. Local versus Foreign Context. A third design concern relates to local versus foreign context information. The use case described in Section 4.2 shows a clear example of using local context in publications and subscriptions. Current context-aware publish subscribe approaches limit the use of context only for entities foreign to the publisher or subscriber. In Clara's case it would be required to modify the Phone components of all those who call her in order to enforce the location restrictions. A Phone component calling Clara would be required to specify that the Phone component receiving the call event would only be notified of the event if its location is different from Clara's home. Similarly, a Phone component receiving a call event from Clara's Phone component would be required to specify that the location of the event publisher be different from Clara's home. Through this example it is clear that Clara's Phone component has no control over its behaviour of placing, accepting or rejecting a call events. It is entirely dependent on external components and their knowledge of a foreign entity's context.

6. Related Work

Works related to our approach fall under two categories: *(i)* context-aware frameworks and middleware and *(ii)* context-awareness in publish subscribe schemes and distributed event-based systems.

Numerous context-aware middleware and frameworks have been proposed. The Context Toolkit [2, 3, 4] is one of the first approaches to a context-aware framework. It applies a graphical user interface approach, where the context widgets are used to provide access to context information. The context toolkit advocates the use of events as an optimal solution to disseminate context information. However, it does not use distributed events and the approach is similar to the observer pattern. SOCAM [5, 6] is a Service-Oriented Context-Aware Middleware that enables rapid prototyping of context-aware services. The SOCAM architecture consists of context providers, context interpreters, context database, service location service and context-aware mobile services. It uses Java's RMI technology to provide component communication and context information dissemination. This approach is also similar to the observer pattern. The Context Broker Architecture (CoBrA for short) [7, 8] is an approach that provides a middleware to create context-aware software agents. CoBrA divides the context model into different domains. Each domain has an autonomous agent, namelly, the domain context broker. Context acquisition in CoBrA is accomplished in the same fashion as in the context toolkit and SOCAM. In all of the systems described above each component is responsible for searching for and registering with a context source. All of the context information is disseminated by the context source directly to the registered components. This approach is different from a distributed event-based system. MoCoA [9] is a customizable middleware for context-aware mobile applications, based on STEAM [10], a distributed event-based system, and the sentient object model [11]. MoCoA introduces the notion of event channels, which are used to specify the constraints on both propagation and notification of events in a geographical area. Even though MoCoA is classified as a distributed event-based system, it does not provide a context-aware event model.

Some approaches deal with context awareness in publish subscribe schemes and distributed event-based systems. In [12] Cugola et al. propose the introduction of context as a first class element in event-based systems. This modification allows each node to set its current context state and allows for subscriptions with both content and context filters as well as publication with context filters. In [13] Frey et al. propose a context-aware publish subscribe scheme. The approach of Frey et al. focuses on limiting the subscription scope through context of relevance and context of interest. Context of relevance is used by a publisher to represent either the context that is affected by the event or the context in which the event is relevant. Context of interest is used by subscribers to identify the context of a publisher. Both the approaches by Cugola et al. and Frey et al. focus on the context of both publishers and subscribers as external

entities to the component. Said context is used to validate if the publisher of the event is relevant to the component with respect to its subscription or, in the case of publication, if the subscriber to the event is relevant. This emphasis on factors external to the component for publish subscribe systems is undesireable since it causes a higher coupling between components. A subscriber must be made aware of specific details of a publisher's context states in order to subscribe to an event.

7. Conclusion

In this paper we have claimed that distributed event-based models and approaches should use context as a first class element, where the local context of its components should be taken into consideration in publications and subscriptions of events. To this end, we have introduced a context-aware event model and a new publish subscribe scheme. In addition we have described a context-aware distributed event-based framework that provides support for mobile context-aware user interactions through real time subscription and publication behaviour adaptation. We have also discussed the utility of our approach through a scenario based evaluation.

As future work, we are working on the implementation and performance evaluation of an object-oriented version of the proposed framework using the Android platform. We will also work on the formalization of our models to be able to support the verification of critical context-aware applications based on the our proposed publish subscribe scheme. Overall, we believe that an approach that combines context and distributed events and supports a publish subscribe scheme that is affected by local context will contribute to further research exploration in mobile and context-aware systems.

8. References

[1] R. Kazman, G. Abowd, L. Bass, P. Clements, Scenario-based analysis of software architecture, IEEE Softw. 13 (1996) 47–55. doi:10.1109/52.542294.
URL http://portal.acm.org/citation.cfm?id=624616.625653

[2] A. K. Dey, G. D. Abowd, D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications (2001).

[3] A. K. Dey, D. Salber, G. D. Abowd, The context toolkit: aiding the development of context-enabled applications, in: Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, CHI '99, ACM, New York, NY, USA, 1999, pp. 434–441. doi:http://doi.acm.org/10.1145/302979.303126.
URL http://doi.acm.org/10.1145/302979.303126

[4] A. K. Dey, J. Mankoff, Designing mediation for context-aware applications (2005).

[5] T. Gu, H. K. Pung, D. Q. Zhang, A middleware for building context-aware mobile services, in: Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th, Vol. 5, 2004, pp. 2656 – 2660 Vol.5. doi:10.1109/VETECS.2004.1391402.

[6] T. Gu, H. K. Pung, D. Q. Zhang, A service-oriented middleware for building context-aware services, J. Netw. Comput. Appl. 28 (2005) 1–18. doi:10.1016/j.jnca.2004.06.002.
URL http://portal.acm.org/citation.cfm?id=1053030.1053031

[7] H. Chen, An intelligent broker architecture for context-aware systems, Tech. rep., University of Maryland (2003).

[8] H. Chen, T. Finin, A. Joshi, Semantic web in the context broker architecture, in: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), PERCOM '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 277–.
URL http://portal.acm.org/citation.cfm?id=977406.978667

[9] A. Senart, R. Cunningham, M. Bouroche, N. O'Connor, V. Reynolds, V. Cahill, Mocoa: Customisable middleware for context-aware mobile applications, in: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Vol. 4276 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 1722–1738.
URL http://dx.doi.org/10.1007/11914952_47

[10] R. Meier, V. Cahill, Exploiting proximity in event-based middleware for collaborative mobile applications, in: J.-B. Stefani, I. Demeure, D. Hagimont (Eds.), Distributed Applications and Interoperable Systems, Vol. 2893 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 285–296.
URL http://dx.doi.org/10.1007/978-3-540-40010-3_26

[11] G. Biegel, V. Cahill, A framework for developing mobile, context-aware applications, in: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), PERCOM '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 361–.
URL http://portal.acm.org/citation.cfm?id=977406.978672

[12] G. Cugola, A. Margara, M. Migliavacca, Context-aware publish-subscribe: Model, implementation, and evaluation, in: Computers and Communications, 2009. ISCC 2009. IEEE Symposium on, 2009, pp. 875 –881. doi:10.1109/ISCC.2009.5202277.

[13] D. Frey, G.-C. Roman, Context-aware publish subscribe in mobile ad hoc networks, in: A. Murphy, J. Vitek (Eds.), Coordination Models and Languages, Vol. 4467 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 37–55.
URL http://dx.doi.org/10.1007/978-3-540-72794-1_3